NAT Router Technical Reference

Interniche Legacy Document

Version 1.00

Date: 11-May-2017 14:02

Table of Contents

1 Introduction

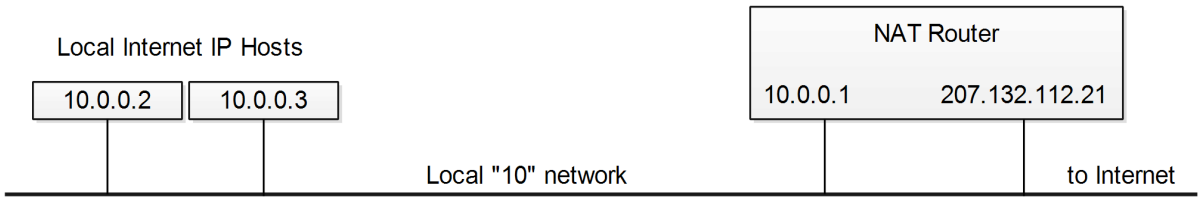
This document is a "how-to" manual to enable an experienced embedded systems programmer, with a conceptual understanding of what a NAT Router does, to port the InterNiche NAT code to router, hub, or similar device.

1.1 NAT Description

NAT allows client IP hosts on a stub network connected to the Internet to access Internet hosts without having to obtain and assign "real" IP addresses for each host. It works by modifying the IP headers IP addresses and selected fields in upper layer protocol headers so that the hidden internal IP addresses are replaced with a "real" InterNic assigned IP address which can safely traverse the Internet. Once the NAT Router is assigned at least 1 "real" IP address, up to 64 thousand IP client machines can share this address to simultaneously access Internet hosts.

This technology is based on Internet standards. The basic NAT was initially described by RFC1631 and subsequently by RFC3022. The class A address "10.0.0.0" (hereinafter referred to as the "10 net") is reserved for private nets, such as those routed by NAT, by RFC1597. The InterNiche NAT code can use any address, but unless there is some legacy addressing to be supported (usually due to switching ISPs) the 10 net is recommended.

A simplified example of a NAT routed IP network is illustrated here:



Let's trace an example packet in this net. In these examples, the IP address/TCP Port number pairs are shown in the format: `Source_IP_address:TCP_Port -> Destination_IP_address:TCP_Port`.

The local machine with IP address 10.0.0.2 wants to browse a page on an Internet Web server 199.1.1.1. This machine is set up, either manually or with DHCP, for its default router to be 10.0.0.1; thus the first packet is sent to 10.0.0.1 for routing. It has the IP addresses and TCP port numbers of 10.0.0.2:1025->199.1.1.1:80. The 1025 could be any unreserved port number, we just picked 1025 for this example.

The NAT Router receives this packet, and by comparing subnet masks, realizes it should be routed out to the Internet. NAT modifies the IP and TCP headers to replace the local 10 net address with its own IP address, and replace the client port number with one not currently in use by the NAT Router. The resultant packet in our example may have numbers `206.86.223.7:2033->199.1.1.1:80`. This packet can traverse the Internet to the Web server without problems.

When the NAT Router makes this conversion, it creates a dynamic table entry which maps the local to Internet address and port numbers. That way when the Web server across the net replies, the NAT Router can reverse the translation process and send a packet back to the 10.0.0.2 host.

1.2 Terms and Conventions

In this document, the term "NAT", when used without other qualification, means the InterNiche NAT routing engine as ported to an embedded system. "System" refers to your embedded system. A "user" or "porting engineer" usually refers to the engineer who is porting the NAT. An "end user" refers to the person who ultimately ends up using the "user's" product.

Names of files, C structures, and C routines are displayed as follows: `c_routine()`.

Source from C programs is displayed in these boxes:

```
/* C source file - yet another 'hello' program. */
main() {
   printf("hello world.\n");
}
```

The term "local" or "private" network refers to a stub network which is using private IP addresses. The term "Internet" or "outside" network refers to the larger Internet (or Intranet) where registered public IP addresses are required. A NAT Router translates many private IP addresses from its local network into one or more public IP addresses for sending on the outside network. On a firewall router which has a third "DMZ" net for local servers which are publicly accessible, the DMZ would be considered an outside net.

2 System Requirements

The InterNiche NAT layers are very lightweight, but they do require some support from the host system to operate. NAT's needs fall into these three classes, each of which is explained in detail below.

- Static memory
- Dynamic memory - malloc() and free()
- Periodic clock tick

2.1 Static Memory

As with all embedded system code, the NAT code takes up some Code and Data space. On embedded systems the code is usually stored in ROM, and may be moved to RAM at boot time. The exact amount of code space required will vary depending on which NAT features you enable through defines, your processor, and your compiler.

2.2 Dynamic Memory

Each time a host on the local net makes a connection to an Internet host, a connection table entry must be built in dynamic memory. For the simple cases where there is no fragmentation, these table entries may be as small as 80 bytes each (as is the case with the above DOS Demo example). They may increase slightly on 32 bit processors due to word alignment requirements.

Note that a single host on the local net can have more than one connection open at once. One of the most connection intensive applications, a Web Browser, reading a typical HTML page with (for example) three embedded graphics, will typically need four simultaneous connections. Thus the dynamic memory requirement estimate to support 50 end users all simultaneously loading web pages might be calculated:

```
50(users) X 4 (connections) X 80 (bytes) = 16,000 bytes
```

Please keep in mind that this is only an estimate; the actual numbers will be affected by everything from the author of the Browser to the complexity of the HTML page being read.

2.3 The Clock Tick

The NAT routing code includes a routine which should be called by the system once a second. This routine is in charge of freeing connection table entries that are no longer being used.

3 Architectural Overview

The InterNiche NAT Router code implements Network Address Translation (NAT) as described in RFC1631. The code is written entirely in ANSI "C". The basic NAT is implemented in three .c files and one .h file, which need to be compiled and linked into the IP code of the target system. The porting engineer needs to provide additional C code to call NAT entry points and initialize variables as described below.

The code has two primary runtime entry points: a single call made from the IP layer when packets are received which are candidates for routing (on most routers this means any IP packet), and a routine called once a second for deleting timed-out table entries.

In addition to these entry points, several static variables need to be set up before NAT can operate. These are:

nat_inaddr	the IP Address of the interface to the Internet
nat_inmask	the subnet mask for nat_inaddr
localnet	the IP address of the interface on the local net (10.0.0.1 in our example)
localmask	the subnet mask for localnet.

Most of these can be set to factory defaults which don't need to be changed by the end user. The sole exception is the Internet IP address, which can usually be set by an automated protocol like PPP or DHCP. They must all be set to reasonable values for NAT to function.

There are also some optional features of NAT which may require setup. These are:

tcp_timeout	the number of seconds to keep idle TCP connections (see TCP Timeout for details)
udp_timeout	the number of seconds to keep idle UDP entries (see TCP Timeout for details)
Server List	A single server of each service (Web, FTP, Telnet) on the local net can be made available to the outside Internet world by putting them in this list.
Alias List	Any IP address on the local network may be translated to any other IP address by making an entry in this list for it.

4 Porting Step byStep

This section outlines the steps needed to port the InterNiche NAT code into the software of an existing IP router. The general assumptions are made that the router's software is written in C language, that the router's system software is linked from C libraries and/or object files, and that the required static and dynamic memory is available.

4.1 Source Files

As provided, the NAT Router is seven .c source files, and four .h (include) files. These are called the NAT "portable" or "port-independent" source" files. These portable files should not need to be modified for a simple NAT port. Two additional files, `natport.c` and `natport.h`, are provided as part of the reference port, and implement the port dependent portions. The functionality of these port dependent files will need to be duplicated in the target system as part of the porting process.

The portable .c source files are:

File	Description
nattl.c	main NAT Translation routine entry point
natutil.c	utility functions used by nattl.c
natfrag.c	instruments fragmentation handling (controlled by compile-time use of `#define IP_FRAGMENTS`)
natftp.c	handles FTP over NAT
nat_nt.c	CLI support for NAT
nat_mod.c	Module support for NAT

The portable .h files are:

File	Description
natip.h	Function prototypes and protocol header structures.
nattab.h	NAT connection, lookup, and hash table structures.
natfrag.h	Structures and settings specific to fragmentation handling.

The reference port files are both quite small - less than 10K bytes of commented source. They are:

File	Description
natport.h	Defines types, maps macros for memory calloc() and free().
natport.c	Sets up NAT variables

4.2 Entry Points

Once you have the NAT code compiling, you need to add the hooks to your existing IP router code to call it. For basic NAT functionality there are only two calls, the actual NAT route call and the once per second timer tick. For additional functionality in an environment with dynamic IP addresses, there is a third call to inform the NAT router that it needs to refresh its view of its IP interface configuration. This section provides an overview of these calls and suggests where they should be called from.

Name

`do_nat()`

Syntax

```
u_char do_nat (PACKET p, PACKET **pktpp, u_short *pkt_countp);
```

Parameters

p	(received) packet to be processed by the NAT module
pktpp	pointer to an an array of PACKET pointers, each of which represents one PACKET that is being returned by the NAT module back to the stack for further processing
pkt_countp	pointer to count of the number of packets in the `pktpp` array

Description

`do_nat()` is the basic NAT translator function. It accepts a pointer to an IP packet to translate.

IP and TCP options are OK - these will be handled by NAT and should not be deleted from the packet. The general rule for `do_nat()` is that the packet should appear in the buffer as is was received from the media.

`do_nat()` determines if the packet should be NAT routed. If, for example, the packet is from a "Real" IP host on the local segment, and bound for an IP host out on the Internet, then conventional IP routing should be done (usually in this type of environment the packet is forwarded to the ISP's default router).

If the packet should be NAT routed, `do_nat()` changes the headers in the passed buffers as required. The returned packet is ready to be sent to the lower (sub-IP) layer for the destination net.

Returns

The do_nat() function can return either of the following two values:

0	Indicates that the NAT module has "consumed" the packet, and that `ip_rcv()` does not need to process the packet any further (and can return immediately).
1	Indicates that one (or more) packet(s) are being returned to `ip_rcv()` for subsequent processing.

Where to put the call to do_nat()

Generally `do_nat()` should be called from the IP layer software after the received IP packets have been verified, but before any data has been lost or altered. If you are using an IP stack derived from Berkeley UNIX, the call should be made from the file `ip_in.c` in the routine `ipintr()` after the received packet has had the IP checksum checked and before the comment "convert fields to host representation". If the packet is NAT routed by `do_nat()` you will need to write code to find the appropriate ifp (interface pointer) and send the packet using the `ifp->if_output()` call. If you are using the InterNiche IP stack, the call to `do_nat()` is already coded in the file `ipdemux.c` inside the `#define NATRT`.

Name

`nat_timeisup()`

Syntax

```
void nat_timeisup(void);
```

Parameters

None.

Description

This is the timer routine used to age out old NAT table entries. This should be called once a second. It requires no parameters.

Returns

Nothing.

Name

`nat_re_init()`

Syntax

```
void nat_re_init(void);
```

Parameters

None.

Description

This function re-initializes the NAT router's IP addresses and IP network masks from current interface values. It is typically provided by the porting engineer, and a sample implementation that is suitable for the InterNiche IP stack is provided in `natport.c`.

Returns

Nothing.

Where to put the call to nat_re_init()

`nat_re_init()` should be called after the IP address or IP netmask on a NAT interface changes. This is usually done from protocols that dynamically assign interface addresses such as DHCP and PPP/IPCP.

4.3 Dynamic Memory Use

Once you have inserted, compiled, and debugged the calls to these two routines, you have to make dynamic memory available to the NAT routing code. All the NAT code's dynamic memory is allocated by calls to `NATALLOC()` and released by called to `NATFREE()`. The syntax for these is exactly the same as the standard C library calls `malloc()` and `free()`, with the exception that buffers returned from `NATALLOC()` are assumed to be pre-initialized to all zeros. In this respect `NATALLOC()` is like `calloc()`.

If your embedded system already supports standard `calloc()` and `free()` calls, all you need to do is add the following lines to natport.h:

```
#define NATALLOC(size)calloc(1, size)
#define NATFREE(ptr)free(ptr)
```

If your system does not support `calloc()` and `free()`, you will need to implement them. An exhaustive description of how these functions work and sample code is available in *The C Programming Language* by Kernigan and Ritchie.

4.4 Values and Lists

Once you have the nat_ calls and dynamic memory allocation working, you almost have a working NAT Router. The only step still remaining is to decide which of NAT's tunable parameters should be set by the programmer, and which can be made available to the end user. This section describes these parameters and makes some recommendations.

Internet Address

This is the IP address of the interface which is connected to the Internet (as opposed to the local net). This is the one parameter in this section that absolutely must be set for the NAT Router to work. Generally this parameter must be set by the end user, either manually when the router is installed, or automatically via a mechanism such as client DHCP or PPP. The internet address is set by setting the variable named internet. This is a static 32 bit variable of type ip_addr.

Local Address

The IP address of the local or "private" net.

TCP Timeout

This unsigned 16 bit number is the time, in seconds, for the NAT routing code to wait before deleting a connection entry which is not being used. The default is currently 300 (5 minutes). Usually TCP connections which are no longer needed or have reached their retry limit are terminated by one of the endpoints. When this happens the NAT routing code detects the connection closing (via TCP `FIN` or `RESET` packets) and cleans up its NAT entry for the TCP connection. The `tcp_timeout` variable is only intended to come into play when both TCP hosts have stopped operating. In that case, there should be some upper limit on the amount of time an entry will be maintained, but it should be much longer than any TCP host would reasonably take to respond, even over a remote network link.

Making it user setable gives the end user the option to conserve memory space in an environment where router memory space is at a premium and there are lots of TCP connections to unstable machines. The risk is that the end user will set it so low that slow live connections will get deleted along with unused ones. In this case retries from the local machines will result in a new entry being built; but retries from the remote machine (after a connection has been deleted from the NAT router's tables) will result in the packet being dropped.

Another risk of setting this value too low is that on heavily used systems, address "churning" can theoretically provide a means of entry into the private network.

Please note that the TCP timeout does not distinguish between inbound or outbound data.

UDP Timeout

This is similar to the `tcp_timeout` described above, however due to the non-connection nature of UDP the NAT code cannot detect the connection going down. Thus the UDP timeout is almost always the mechanism for deleting UDP connection entries. The default value of 120 (two minutes) reflects this, being much lower than the TCP timeout.

Server List

Also known as the "proxy" list since a server set up in this list has similarities to a proxy server. It is accessed by the pointer variable `proxyList` defined in the file `nattl.c`. For each TCP or UDP port value, one machine on the local net can designated as a server. Packets from the Internet (outside) interface, which would normally be dropped, are instead NAT routed to the server. This allows the end user to set up one server of each type (FTP, Web, Telnet, etc.) on his local net and make them visible to the outside world.

If this feature is desired, the list should be filled in from non-volatile ram or from a file at init time. For example code, see the file `natport.c`.

Alias List

The alias list is designed to allow any IP address on the local net to be translated to a different IP address when it is sent to the Internet. Normally the NAT routing code determines which IP addresses are suitable for NAT routing to the outside by checking the network numbers of the packet's IP addresses. A packet from the local net with a non-local network number would be ignored by the NAT routing code, although regular IP routing might later forward the packet to the outside without altering its IP addresses. The alias list provides the end user with a way to force certain addresses to always be translated.

This feature was provided to satisfy requests from customers which had changed ISPs. The customer's old ISP had assigned them a set IP addresses, which they had configured into all their computers. The new ISP assigned had different IP addresses, and the customer wanted to avoid the work of changing all the addresses in all the machines. By setting up an alias list they can map the old IP addresses to the new ones, or to the NAT router's outside IP address.

The alias list is accessed by the variable `aliasEntries` which is defined in the file `nattl.c`. If this feature is desired, the list should be filled in from non-volatile ram or from a file at init time. For example code, see the file `natport.c`.

Filter or Security Lists

Please refer to the chapter on `IP Filtering` in the `NicheStack Technical Reference Manual` for information pertaining to the creation and maintenance of filtering lists.

NAT with IPSec

The NAT Router is capable of providing a pass through of IPSEC, ESP (Encapsulation Security Payload), and AH (Authentication Header) traffic for a SINGLE IPSEC HOST AT A TIME from the local network to the outside network. This functionality can be enabled by defining the macro `IPSEC_WA` in the `ipport.h` file while compiling the NAT Router.

To ensure that an existing IPSEC session completes predictably, the global `nat_ipsec_local` should be set. This variable is configurable via the `natset -f` CLI command.

If the value is not configured then a warning message is printed while adding a new entry in the NAT Connection list, as another IPSEC host on the inside can break an existing IPSEC session.

Dynamic assignment of `nat_ipsec_local` will allos multiple hosts on the inside can be serviced reliably, but only one at a time.

Fragmented Packets

The passing of fragmented packets through the NAT router is done with the assistance of an internal structure which stores information contained within the header of the packet's initial fragment. These `nat_frag_entry` structures remain active until either all fragments have been passed or `NFE_TMO` seconds have elapsed since the first fragment was received. `NFE_TMO` is defined in `natrt/natfrag.h`

To reduce the likelihood that a denial-of-service attack could result from transmission of fragments to the NAT, the maximum number of bytes held by the NAT while it is waiting for the initial fragment is limited by `NAT_MAX_MEM` (defined in `natrt/natfrag.h`). Incoming fragment causing the memory usage to increase beyond this value will be dropped.

5 Command Line Utilities

The NAT package includes several CLI utilities to assist in configuration and verification of your integration of the module into your application. These commands are dependent on whether `INCLUDE_CLI` is defined in `ipport.h`.

5.1 nat natset

Command Name

```
nat natset - configure NAT module
```

Syntax

```
natset {-a -b <inside IPv4 address> -c <outside IPv4 address>} -e <0 | 1> -
f <IPv4 address> -i <inet network #> -l <local network #> -m <MSS, bytes> {-
o -p <protocol, TCP or UDP> -q <inside port #> -r <inside IPv4 address> -s
<outside port #>} -t <TCP timeout, seconds> -u <UDP timeout, seconds> -w
<TCP window, bytes>
```

Parameters

`-a`	Create alias list entry.
`-b`	Inside IPv4 address for alias list entry.
`-c`	Outside IPv4 address for alias list entry.
`-e`	Enable (or disable) NAT.
`-f`	Set local IPsec host's IPv4 address.
`-i`	Set inet network number.
`-l`	Set local network number.
`-m`	Set TCP MSS (bytes).
`-o`	Create proxy server list entry.
`-p`	Protocol (TCP or UDP) for proxy server list entry.
`-q`	Inside port number for proxy server list entry.
`-r`	Inside IPv4 address for proxy server list entry.
`-s`	Outside port number for proxy server list entry.
`-t`	Set TCP timeout (seconds).
`-u`	Set UDP timeout (seconds).
`-w`	Set TCP window size (bytes).

Description

This command sets various configuration parameters for the NAT module.

Location

This command is provided by the NAT module when NATRT is defined.

5.2 nat natxip

Command

`nat natxip - expunge IPv4 address from NAT tables`

Syntax

`natxip -a <IPv4 address>`

Parameters

`-a`	IPv4 address to be expunged.

Description

This command deletes entries that correspond to this IPv4 address from the various NAT tables.

Location

This command is provided by the `NAT` module when `NATRT` is defined.

5.3 nat netstat

Command Name

`nat netstat - display NAT statistics and status`

Syntax

```
netstat -a -c {-d -e <outside port #>} -f {-g -i <NAT fragmentation entry
index>} -l -p -s
```

Parameters

`-a`	Display all information.
`-c`	Display NAT connection table.
`-d`	Display NAT connection entry.
`-e`	Outside port number (for NAT connection entry).
`-f`	Display NAT fragmentation queue.
`-g`	Display NAT fragmentation entry.
`-i`	Index of NAT fragmentation entry.
`-l`	Display NAT alias list.
`-p`	Display NAT proxy server list.
`-s`	Display NAT statistics.

Description

This command displays statistics for the NAT module.

Location

This command is provided by the `NAT` module when `NATRT` is defined.